A lot of valuable hints for Programming Assignment 6:

The following figures and steps should help you to write your code for the insert function for Assignment 6. These steps assume that you already have good understanding of Trie data structure and you have understood the codes we have discussed in the class. It also assumed that you have read through the assignment description, and you know what you need to do.

1. Make sure you have attended the Trie lecture (if not watch the recording)
2. Make sure you have attended the lecture where we have discussed the codes (if not, review that)
3. Review the lecture notes and the uploaded code
4. Now, read through the entire assignment description
5. Read the hints provided at the end of the assignment description
6. Now read and observe the following figures to understand how to design your insert function.
7. Also, keep the printAll function for debugging your trie/to see the status of your trie

A node stores:

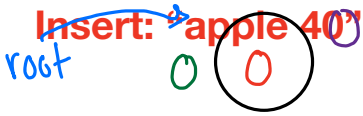
- freq; //similar to count
- sum_freq; //The sum of frequencies for which this string is a prefix of words in the dictionary, including itself.
- cur_max_freq; //current maximum frequency of any child node. It stores maximum sum_freq within this node's 26 children
- * next[26]; //node pointer for 26 children

***(I will use the corresponding color shown above to represent those values in the figure):***

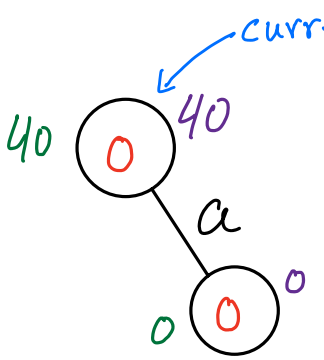
**Working for the first node (Most of them are also applicable for the other nodes too).**

- freq is zero by default. It will remain 0 as we have not reached to complete word "apple" yet.
- Increase sum_freq based on count
- As child 'a' is null, create a node and point that node from 'a'.
- *Now you need to calculate cur_max_freq. But as we don't know the children's sum_freq yet, we calculate ahead what would be the sum_freq by adding count to the existing sum_freq of the child node in that path. So, it is 0+40 = 40. Based on that, decide whether you need to update the cur_max_freq. As 40>0, update it to 40*
- Use the similar approaches for the following steps too!

purple: sum_freq
green: cur_max_freq
red: freq



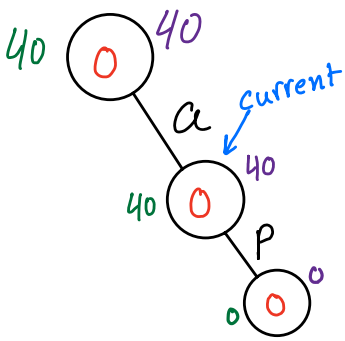

Since this is the first insert, root is only still a default node with no children




apple

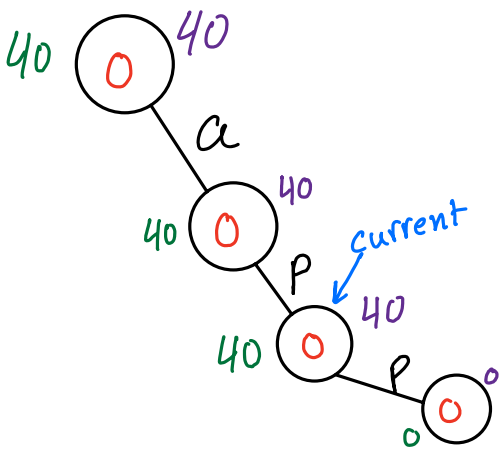
1) - add 40 to prefix sum of current node (0+40=40)
- insert a if it's null (it is)
- update child max to 40 (since 40 > 0)

next child's current prefix sum + count > current max



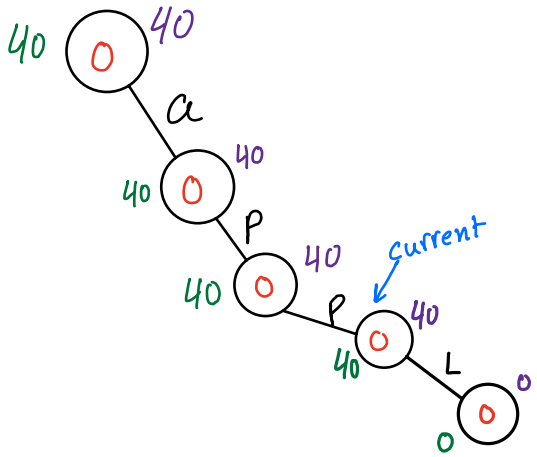

apple

2) - add 40 to prefix sum of current node (0+40=40)
- insert p if it's null (it is)
- update child max to 40 (since 40 > 0)




apple

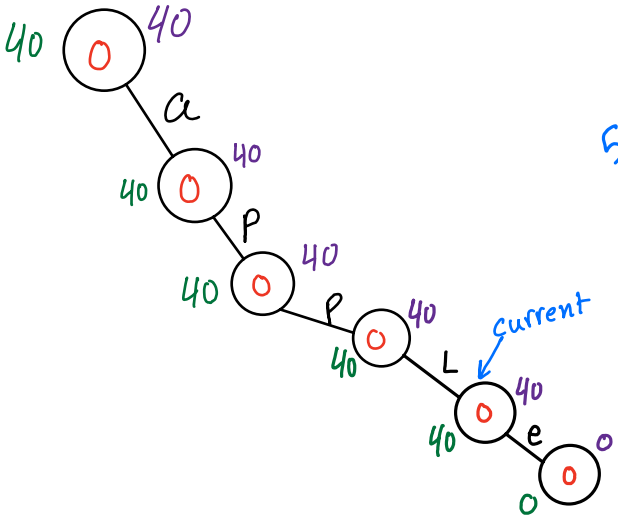
3) - add 40 to prefix sum of current node (0+40=40)
- insert p if it's null (it is)
- update child max to 40 (since 40 > 0)




apple

4) - add 40 to prefix sum of current node (0+40=40)

- insert l if it's null (it is)
- update child max to 40 (since 40 > 0)




apple

5) - add 40 to prefix sum of current node (0+40=40)

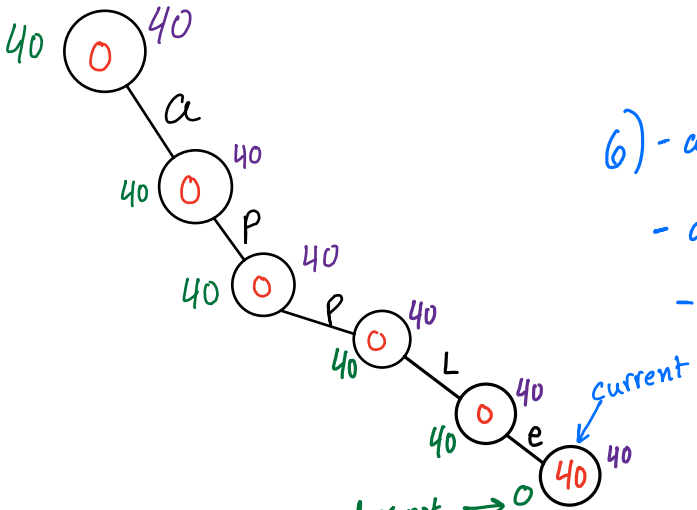
- insert e if it's null (it is)
- update child max to 40 (since 40 > 0)

end: base case

apple

6) - add 40 to prefix sum of current node (0+40=40)

- all letters inserted, updat count (0+40=40)
- stop



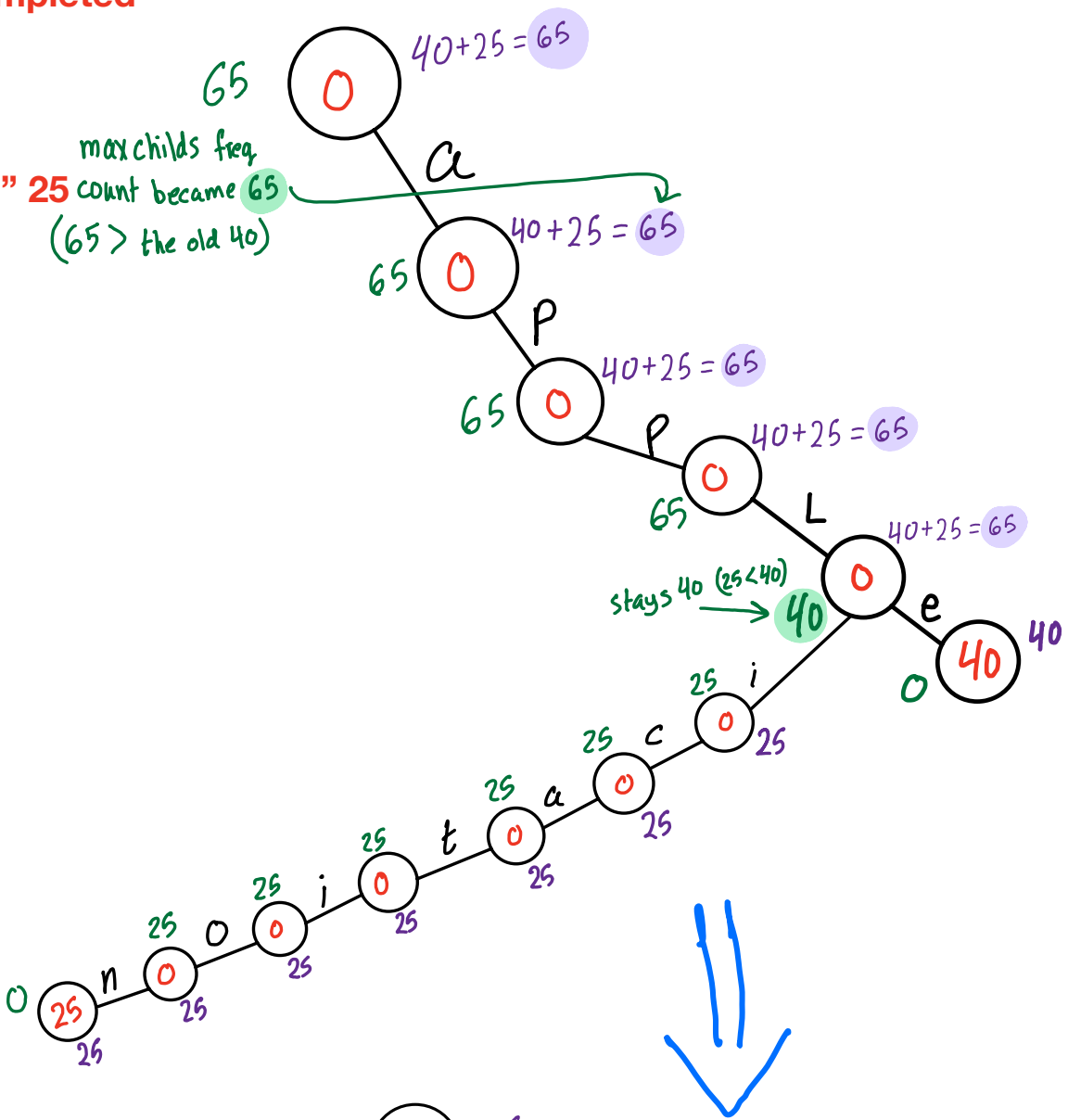

does not → get updated since no child will be inserted (we return before even trying to update it)

Trie #1

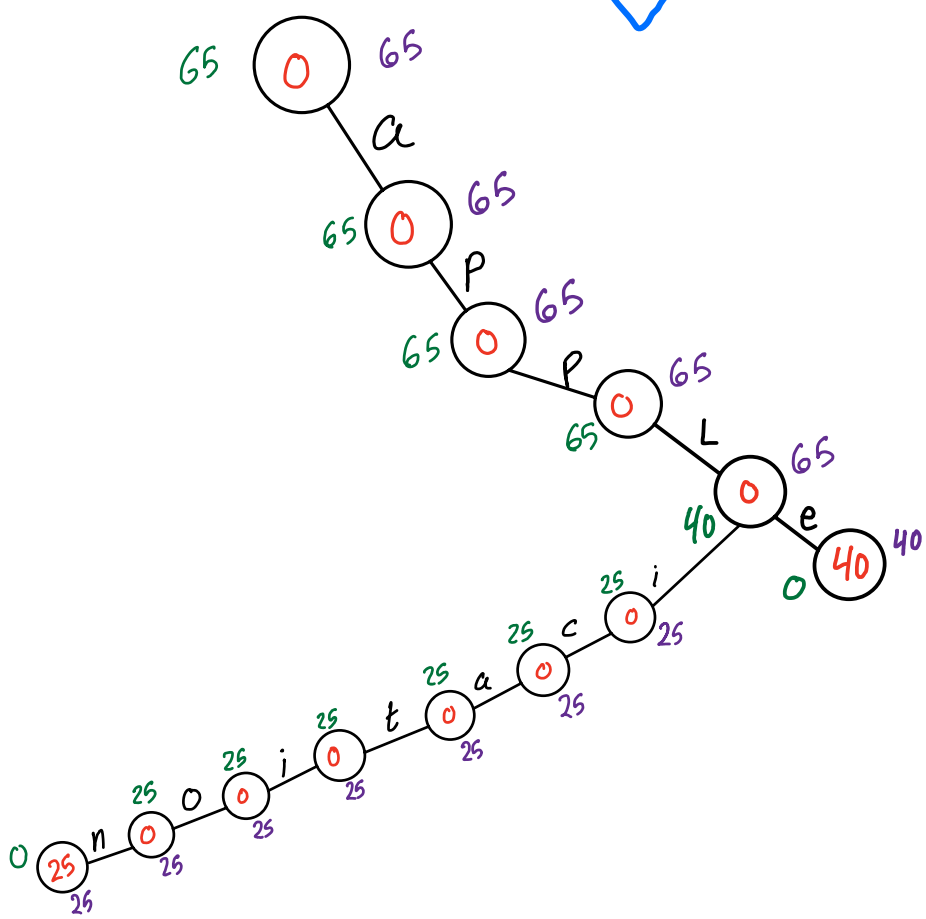
apple insertion completed

Insert “application” 25

max childs freq count became 65 (65 > the old 40)

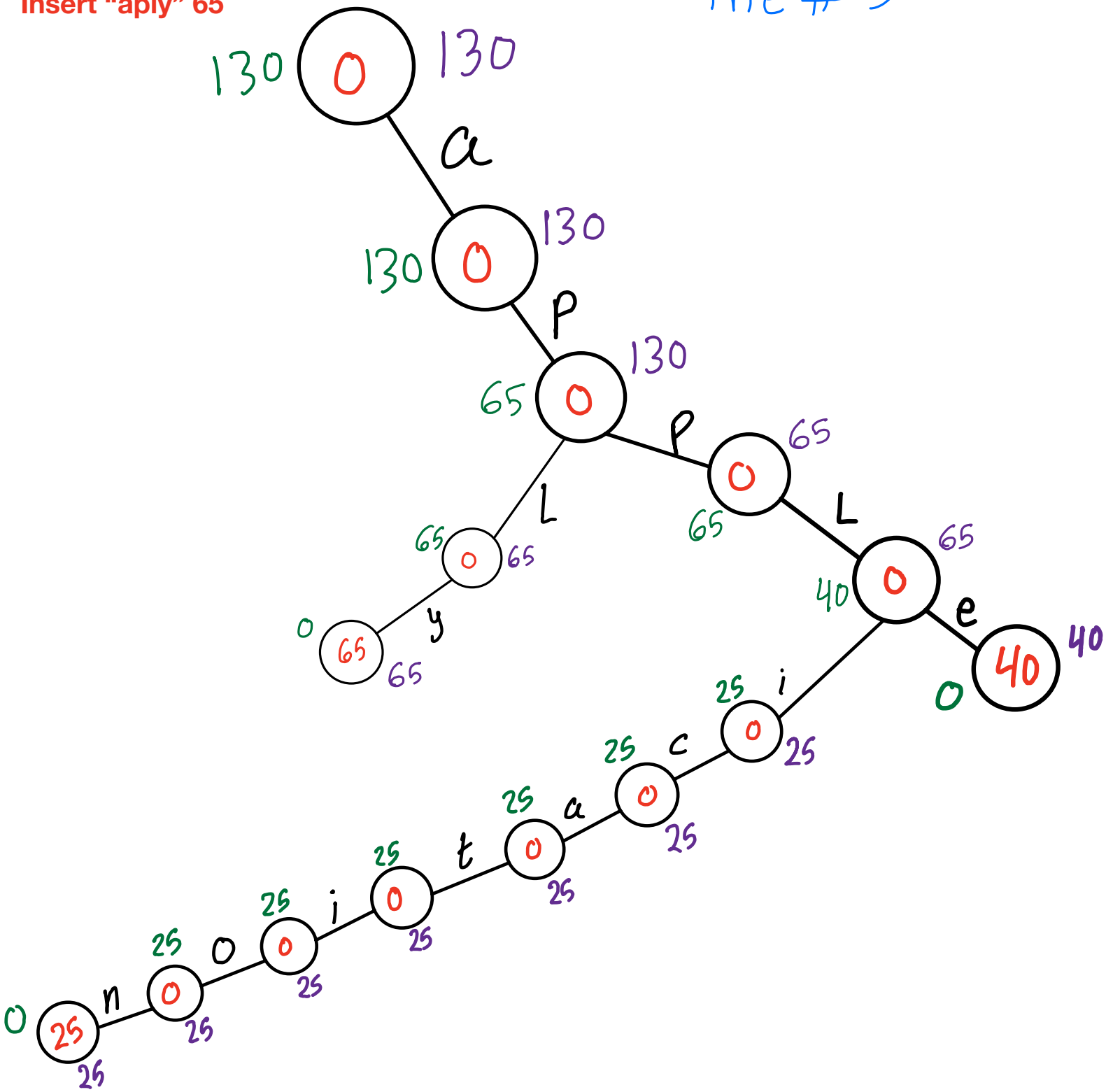



Trie # 2



Insert “aply” 65

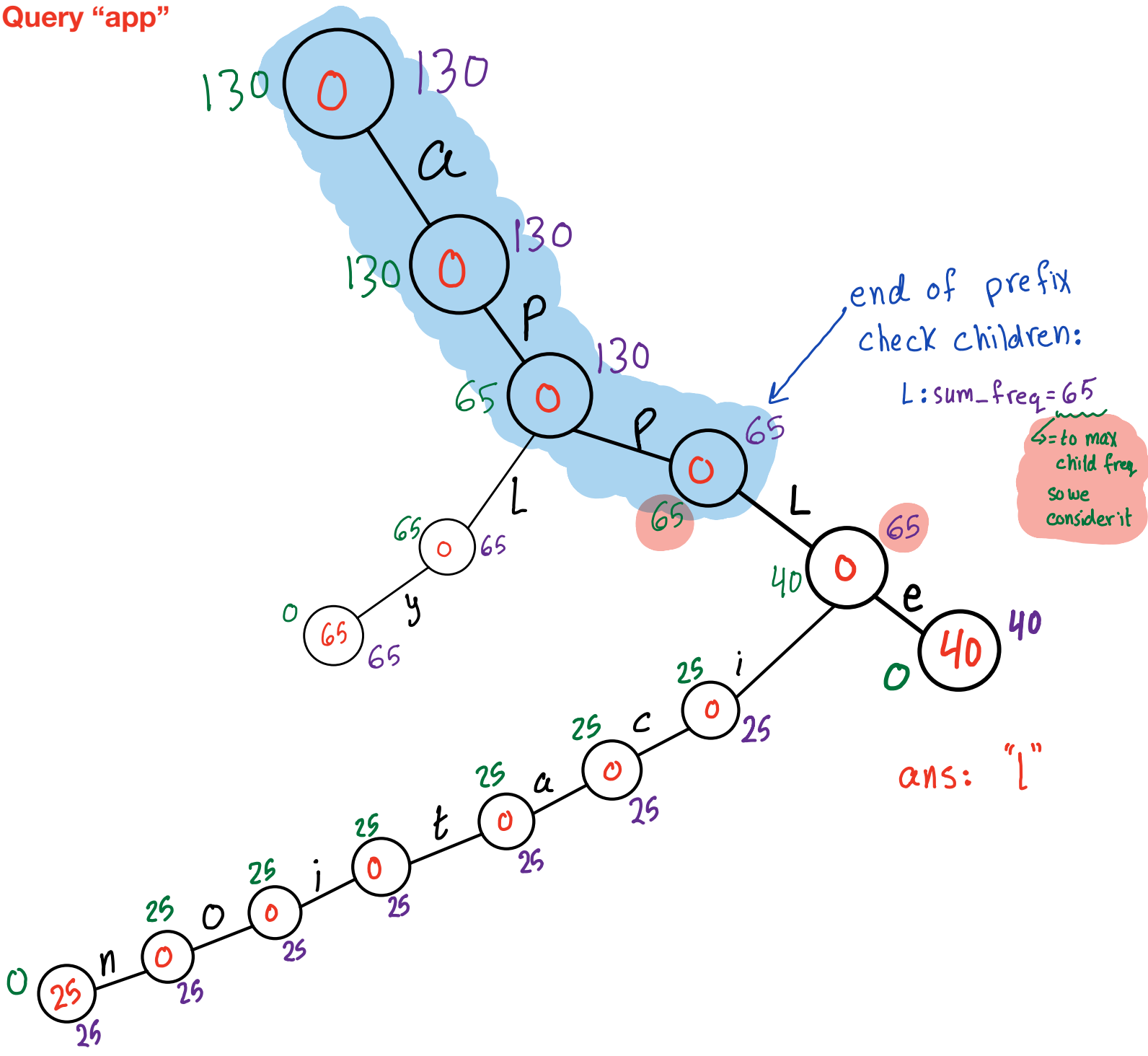
Trie # 3



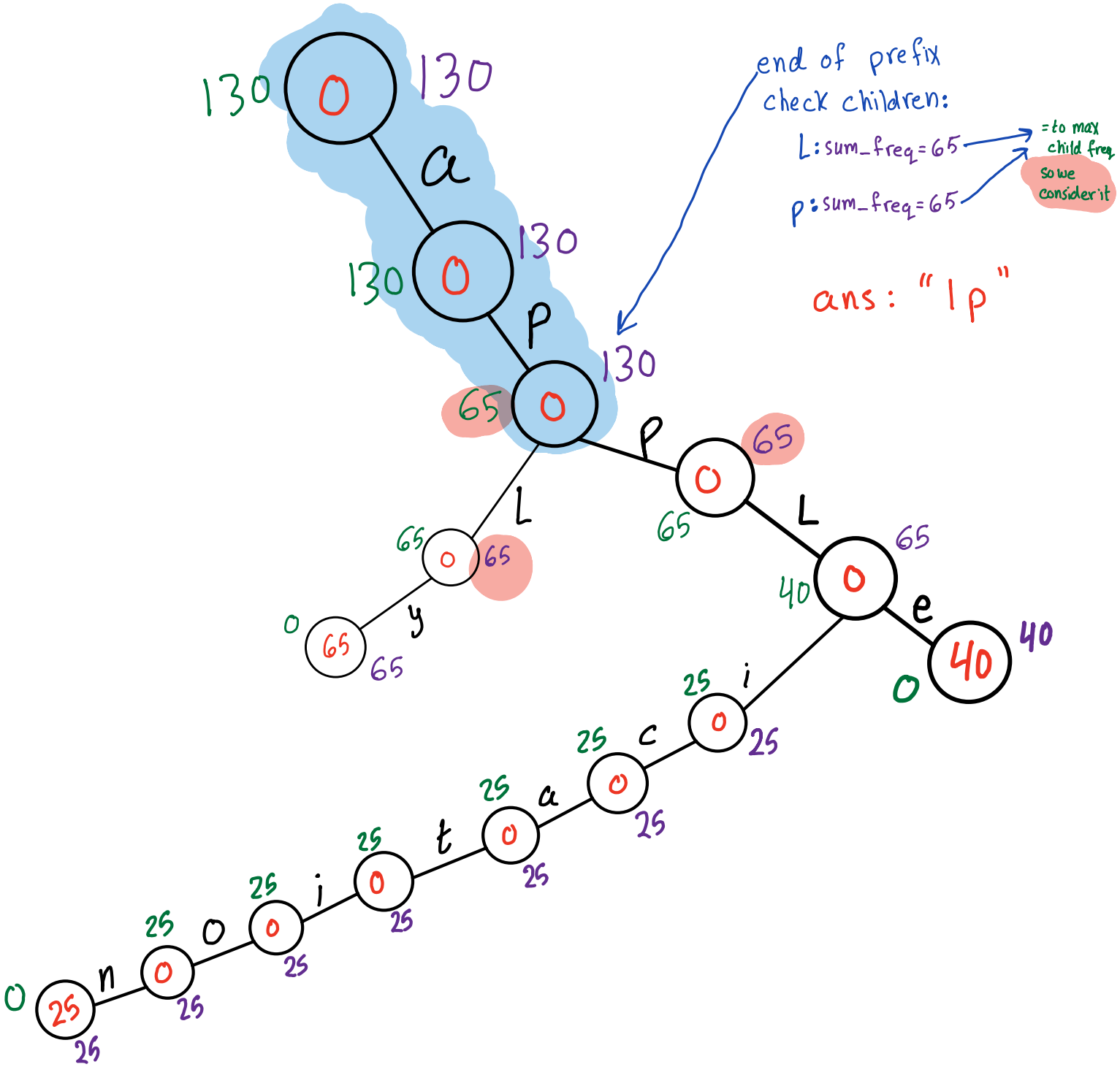
## Query Logic

- Traverse the path using the search logic until you reach the end of the prefix (base case)
- If at any point in traversing, you find that next child does not exist, then prefix is not recognized
- Once you reached the end of the prefix, you need to decide which next child(ren) is most likely to appear, we make this decision by looking at which children have **sum_freq** matching to the current root's **cur_max_freq**
- If multiple children meet this condition, we consider them all
- If node (of last letter in prefix) does not have any children, that's an unrecognized prefix since no letters come after it

**Query "app"**



**Query “ap”**




**Note:**
**these examples had children that all met the (child’s prefix frequency equals current node’s max child freq) condition so we considered them all, if a child does not meet this condition we simply skip it**